BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

EXHIBIT A

Title of disclosure (in English)

Method for Converting Applications using Embedded SQL for Database Access to Portable ODBC Database Access

Idea of disclosure

1. Describe your invention, stating the problem solved (if appropriate), and indicating the advantages of using the invention.

Currently many applications use a relational database as a persistent data store. These applications are often written using embedded SQL to support a single database management system, such as Oracle's ProC. Each database manager has a slightly different implementation of the embedded SQL language used within an application program. This poses a problem when there is a need to make the application work with one or more of the other relational database management systems.

Our invention addresses this problem by converting the applications written using embedded SQL such that the platform independent ODBC interface is used throughout the application instead of the embedded SQL. ODBC is portable across database management systems with little or no change to the application.

The major advantage of our application is that the translation from embedded SQL to ODBC is portable across the database management systems. It results in a single set of application code that is portable, rather than a different set of code for each database management system.

2. How does the invention solve the problem or achieve an advantage,(a description of "the invention", including figures inline as appropriate)?

Our conversion tool consists of two parts, the Source Code Translator, and the Database Access Library of ODBC functions.

The Source Code Translator:

DB2Trans is a tool that is used to create *.c files suitable for access to an ODBC enabled database management system.

DB2Trans will be written using Java because of the rich string manipulation class library in the JDK. DB2Trans will parse *.sql files, containing Oracle Pro C embedded SQL syntax, and produce code that when compiled will make proper calls to the DBAccess module.

The following classes are defined:

- **SmartIn**: This manages the input file, which will be the Oracle *.sql file containing the embedded Pro C SQL calls. A method should be added, called readSQLstmt, that should read one or more lines until the ';' in encountered. This is because most EXEC SQL statements span multiple lines within the editor (CR/LFs embedded) but DB2Trans will just deal with and parse the EXEC SQL

statement as one entity. There should be one public data member which is the 'String inputStatement'. 'InputStatement' will be passed to Statement class for parsing and processing.

- **SmartOut**: This manages the output file, which will be the *.c file incorporating business logic with calls to DBAccess for access to a relational database via ODBC. The SmartOut class from PVSim should be reused. There should be one public data member which is the 'String outputStatement'. There should be methods within SmartOut to add text to this statement based on decisions made in Statement class. The data member outputStatement is what is written to the *.c file for later compilation
- **Statement**: This is where the bulk of the logic will reside. Methods of the classes include:
  - Determine type of SQL statement
  - Add header file
  - Process statement
  - Process nonSQL statements
  - processBeginDeclare
  - processConnect
  - processCommitRollback
  - processSelectInto
  - processPrepare
  - processDeclareCursor
  - processOpenCursor
  - processCloseCursor
  - processFetch
  - processInsertUpdateDelete
  - processExecute
  - processNonTranslatedSQL
  - parseSQLStatement
  - BuildHostVarStructure
  - Make calls to SmartOut to build result string into outputStatement

DBAccess:

DBAccess.c will be a single standard ANSI C module with multiple functions. All DB2 CLI calls will be contained within this module.

The environment and database connection handles will be reused throughout the life of the executable, statement handles will be allocated/freed per statement.

Functions Definition and Logic

```
sqlcode = DBAconn(&user, &pw);
```

This function is used to allocate handles and make the initial connection to ODBC.

sqlcode = DBAterm();

This function is used to make CLI calls to free connection resources, after this processing the executable will no longer be connected to the database and no further ODBC calls can be made. This function is called from DBAuow when release = 1, yes.

sqlcode = DBAuow(int statementtype, int release);

This function will just be a switch statement on the parameter statement type. Obtain the environment and database connection handles, SQL_COMMIT and SQL_ROLLBACK are used depending on the request. Issue the ODBC call SQLTransact(). If Release = 1 (Yes) then call DBAterm which will make ODBC calls to free connection resources, after this processing the executable will no longer be connected to the database and no further ODBC calls can be made.

sqlcode = DBASelectInto(&stmt, &starting address of structures containing data types, returned host variables, and indicator variables)

This function is used to process SQL SELECT statements.

The parser generates a statement in which values have been substituted for all of the host variable required for processing the statement, for example, the value of CARTREFE is known before the parsing begins and is substituted into the SQL string that will be processed.

EXAMPLE: SELECT CEMPRESA, CARTREFE, XXXXXX FROM T6198000 WHERE CARTREFE = '00000000000001'

sqlcode = DBAPrepareDeclare(&stmt)

This function is used to process SQL DECLARE and PREPARE statements.

The parser generates a statement in which values have been substituted for all of the host variable required for processing the statement, for example, the value of CARTREFE is known before the parsing begins and is substituted into the SQL string that will be processed.

EXAMPLE: SELECT CEMPRESA, CARTREFE, XXXXXX FROM T6198000 WHERE CARTREFE = '00000000000001'

sqlcode = DBAFetch (&hstmt, &starting address of

structures containing data types, returned host variables, and indicator variables)
/* This statement is called when an EXEC SQL FETCH is encountered. */

```
sqlcode = DBAEndSelect (&hstmt);
```
/* This statement is called when an EXEC SQL CLOSE CURSOR is encountered. */

```
sqlcode = DBAupdate(&statement)
```
This call will support searched (not cursor related) INSERT/UPDATE/DELETE.

The parser generates a statement in which values have been substituted for all of the host variable required for processing the statement, for example, the value of CARTREFE is known before the parsing begins and is substituted into the SQL string that will be processed.

EXAMPLE: SELECT CEMPRESA, CARTREFE, XXXXXX FROM T6198000 WHERE CARTREFE = '00000000000001'

Supporting Code Design

Host Variable Definition Structure

A structure will be used to contain information needed to process and return data for each host variable used. The structure is defined in db2da.h. The structure will be dynamically allocated from within the processed files, with an instance of the structure per host variable. A single variable containing the address of the start of the list of repeating structures will be passed to the appropriate DBAccess functions.

DB2Trans will do the following:

- When a EXEC SQL BEGIN DECLARE is reached, build a hash table (keyword = host variable name) for each host variable containing type, length, pointer. If a new definition of the host variable is encountered, the old definition is replaced.
- When an SQL statement is encountered, parse the host variables and keep count of the number.
- Host variables are always processed by numeric order, from left to write.
- void *phostvarlist = malloc(number of host variables * sizeof(hostvarstruct))
- Pass into DBAccess hostvar.

The structure definition is:

```
struct  hostvarstruct {
        int datatype;
        int length;
        void *phvlocation;
        void *pindvarlocation;
 };
```

Header file db2da.h

Contents are as follows:

- sqlcode
- pHenv (contain environment handle, reused across exe life)
- pHdbc (contain database connection handle, reused across exe life
- sqlca structure
- hostvarstruct
- #defines for SQL datatypes
- #define for schema name (like 'U604')
- #define for CLI database name (like 'storeflow32')

Runtime Configuration

It is not expected that DBAccess will need any runtime configuration files. All attributes should be bound at build time. The ODBC interface itself can be configured to establish most of the key runtime attributes.